Scaling Strong Consistency Across Continents with Hierarchical Consensus

Omitted for review

Abstract

We introduce Hierarchical Consensus (HC), an understandable implementation and extension of Vertical Paxos designed to create large, strongly-consistent geo-replicated distributed systems. HC uses partitions across multiple independent subquorums to increase overall availability, localize decision-making, and improve throughput. Subquorums can be lightweight and chosen from hosts with low mutual communication latencies such that co-location and balanced load allow them to reach decisions quickly. Globally, a root quorum maintains the decision-space and provides linearizable guarantees across the entire system by intersecting with subquorums using a novel delegation approach to voting. We demonstrate HC's advantages through an implementation scaling to hundreds of replicas across more than a dozen availability zones around the world using Amazon EC2.

Introduction

Coordination to ensure consistent behavior is essential to large-scale distributed software systems, particularly modern systems that span globe [3, 4, **?**, **?**]. A range of mechanisms exist to provide varying levels of consistency, but for systems that require strong guarantees, distributed consensus algorithms are the primary technique for maintaining a correct global state across independent machines in a computing cluster. Consensus design traditionally considers primarily the requirement of fault tolerance: progress in the face of one or two node failures. Current approaches [10, 11, 2, 1] therefore assume a small number of replicas in a co-located group, each centrally located on a powerful and reliable host. As a result, to scale modern systems that also have demanding requirements for high throughput, low latency, and durability, consensus is typically used in layers to manage smaller subsets of the system, trading performance for reliability where possible.

Systems that implement many small quorums of coordination [4, **?**, 3] avoid the centralization bottleneck and reliability concerns of master-service systems [**?**, **?**] but create silos of independent operation that are not coordinated with respect to each other. Recently there has been increasing interest in dynamic and flexible quorums to maintain decentralized ledgers [**?**] but practical experience shows that this results in the instantiation of a large number of intersecting quorums, which is not practical for a dedicated system that does not have to worry about byzantine failure.

Most consensus algorithms have their roots in the Paxos [6] algorithm, originally described in parliamentary terms. The metaphor of government still applies well as we look at the evolution of distributed coordination as systems have grown to include large numbers of processes and geographies. Systems that use a dedicated leader are easy to reason about and implement, however, like chess, if the leader goes down the system cannot make any progress. Simple democracies for small groups solve this problem but do not scale, and as the system grows, it fragments into tribes. Inspired by modern governments, we propose a representative system of consensus, such that replicas elect leaders to participate in a root quorum that makes decisions about the global state of the system. Local decision making, the kind that effects only a subset of clients and objects is handled locally by subquorums as efficiently as possible. The result is a a hierarchy of decision making that takes advantage of hierarchies that already exist in applications.

In this paper, we describe *Hierarchical Consensus (HC)*, an implementation and extension of Vertical Paxos [7]. Like Vertical Paxos, HC reasons about consistency across all objects by identifying commands with a grid ordering (rather than a log ordering) and is reconfigurable to adapt to dynamic environments that exist in geo-replicated systems. Adaptability allows HC to exploit locality of access, allowing for high performance coordination, even with replication across the wide area. HC extends Vertical Paxos to ensure that intersections exist between the subquorums and the root quorum to ensure coordination exists between subquorums and to ensure that the system operates as a coordinated whole. To scale

the consensus protocol of the root quorum, we propose a novel approach, *delegation*, to ensure that all replicas participate in consensus but limit the number and frequency of messages required to achieve majority. Finally, we generalize HC from primary-backup replication to describe more general online replication required by distributed databases and file systems.

The rest of the paper is organized as follows: we first describe the background context that led to the intuition and motivations for HC, primarily describing Raft [12], our consensus algorithm of choice and a vertical consistency model. We then describe the design of the system in detail and describe failure-free operation of HC in a normal environment. We follow the failure-free description with a description of how HC handles various forms of failure and make an argument for the safety of the protocol, including several specific cases of failure. Finally we evaluate the performance of HC implemented as a distributed key-value database, distributed across 15 geographic regions around the world.

Background

Paxos [5, 2, **?**, 11, 1] has two primary optimizations - fast path/slow path and leaders.

We implement and extend Vertical Paxos [7] but go beyond primary backup replication to creating a key/value store (and soon a file system).

Consider Niobe [9] and Boxwood [8] as two implementations of Vertical Paxos.

Raft Consensus

A lightweight description of Raft and leader-oriented consensus protocols.

Vertical Consistency Model

A command is identified by `(Epoch, Term, Log Index)` – mirrors Vertical Paxos `(Configuration, Ballot, Index)` structure.

Consistency: there must be a single, externalized ordering across all objects.

Figure: grid ordering consistency. **PJK: ??**

Related Work

- ePaxos [11] is best geo-replicated consensus, doesn't scale.
- MDCC [4], Spanner [3] are big systems across multiple data centers but isolate objects into tablets that aren't coordinated. **PJK: In what sense? Does MDCC not re-configure tablets?**
- Niobe [9] and Boxwood [8] are Vertical Paxos for primary replication but treat configuration as an independent master quorum and have independent object management.

HC Goals/Claims/Intuitions

consistency claims: We claim that for a current epoch, E_i, this externalized ordering[1] exists for all epochs $\leq E_{i-j}$ such that $j \geq 1$ and E_{i-j} is *closed*, e.g. a tombstone has been written to the logs of all subquorums, S_{i-j}. In the current epoch, E_i it is not possible to externalize a complete ordering, however it is possible to describe an observable sequential ordering of all objects.

performance claims: we should make some.

Design

[2]

Preliminaries: a system is composed of a set of replicas, R, initially we assume that this set is fixed, later we discuss adding or removing to this set. The system self organizes into a root quorum. The root quorum assigns replicas to zero or one subquorums.

During operation, a Replica can be *one* of the following.

1. A follower in a subquorum
2. A leader of a subquorum and a member of the root quorum
3. A hot spare

Replicas that are leaders can simultaneously be a subquorum leader and a root quorum leader. Later we relax this so that any replica can be a member of of the root quorum.

Another simple example

[1] **PJK: Hmm.. Externalized implies linearizability. Accesses will observe this, but that does not mean that we can rebuild it later on. This comment seems to be more true for SC.**

[2] **PJK: I'd make this a straw-man design, along the lines of SUNDR. We can then add delegation and generalized delegation later.**

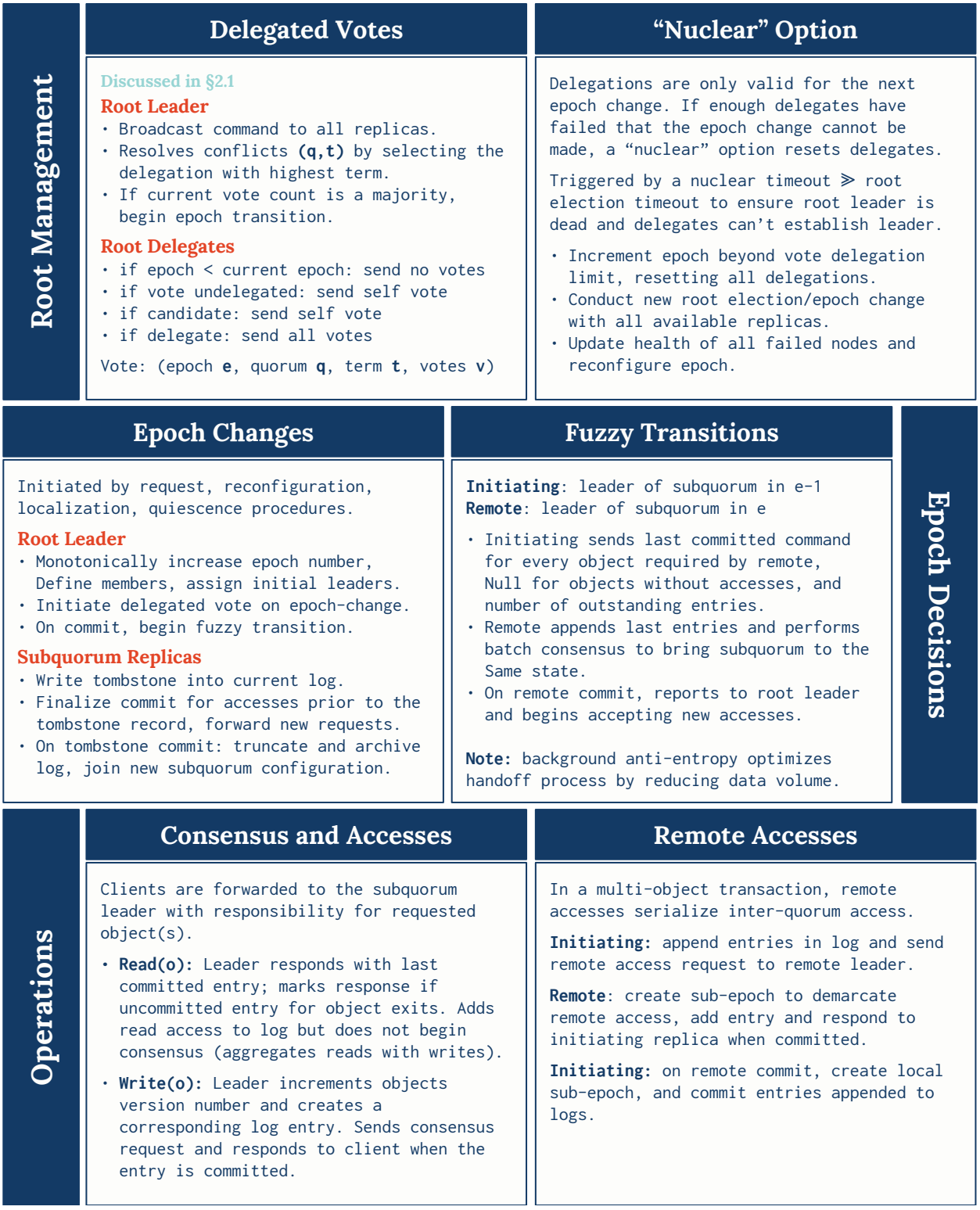

Figure 1: A condensed summary of the hierarchical consensus protocol. Operations are described in a top-to-bottom fashion where the top level is root quorum operations, the bottom is subquorum operations, and the middle is transition and intersection. Section numbers such as 5.2 indicate where particular features are discussed.

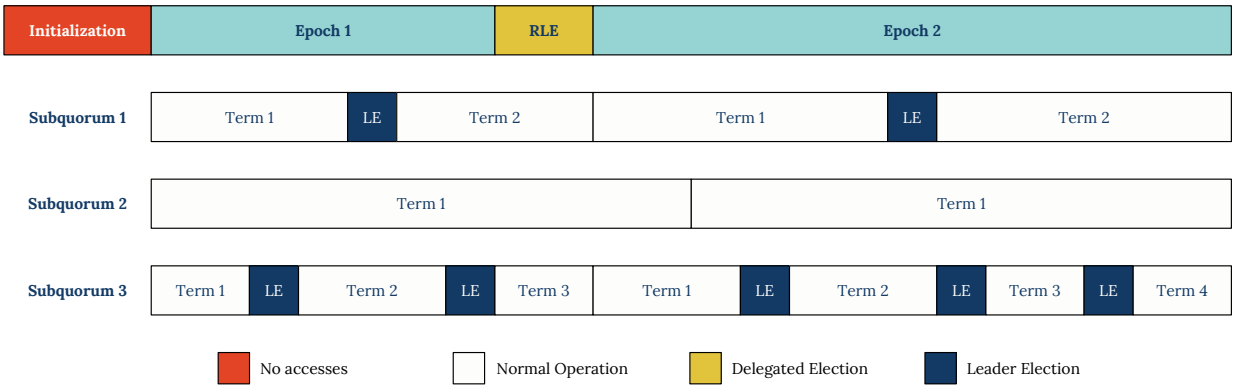

Figure 3: Simple Example: Epochs and Terms Timeline

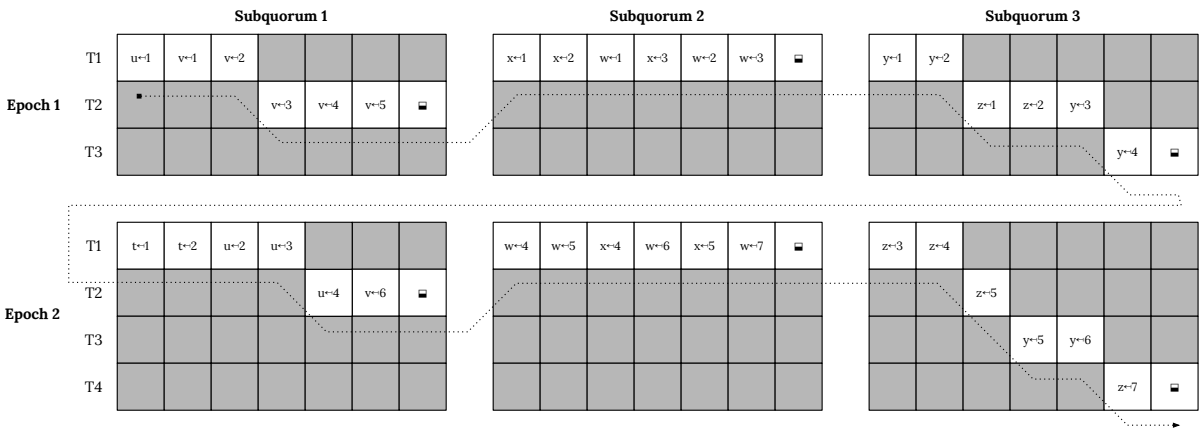

Figure 4: Simple Example: Log Ordering

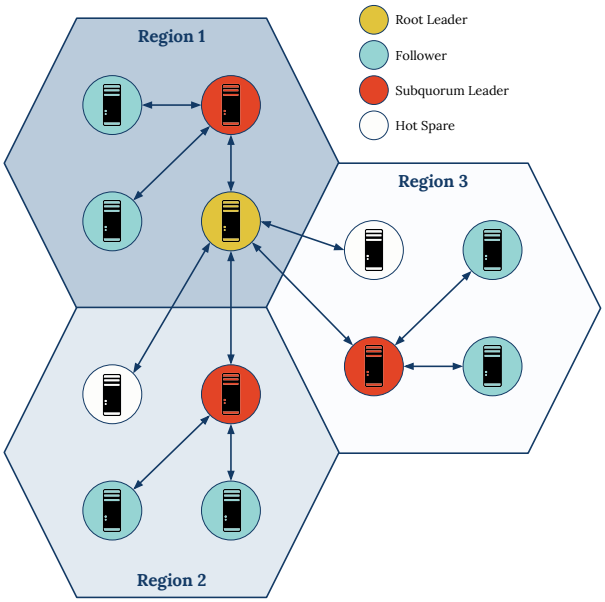

Figure 2: Simple Example: 12x3 HC Topology

Elections and Delegation

- election of root quorum leader
- assignment/election of subquorum leaders
- delegation of votes
- re-elections
- delegations expire

Data Model

BJB: maybe move this to a different location?

- key/value store
- read then write updates **PJK: We assume all writes implemented w/ read then write, or do we implement writes this way? And to what purpose?**
- accesses: get, put, delete

Operation/Decision Making

- subquorum decision making
- remote reads and writes
- reconfiguration/epoch changes
- root quorum decision making
- transitions/fuzzy epochs

Safety and Fault Tolerance

- individual faults
- partitions of the network (big faults)
- safety argument

Delegation Safety

We have implemented two mechanisms for delegation safety: the first is timeout based, and the second is a simple book-keeping mechanism.

In the timeout-based delegation, a replica delegates it's vote for a term limited logical period, usually at least two epochs. This means that the delegated vote lasts through at most 1 root leader election. The nuclear timeout option is longer than the root election timeout, therefore in order for the nuclear option to occur at least one failed root leader election also must have occurred. Because the nuclear option is triggered after a root leader election the delegation is reset on the nuclear option, and every replica must vote for themselves (ensuring that the nuclear option contains no delegations at all).

PJK: Again, I think this fails if delegation can be used for root votes In the book-keeping mechanism, replicas can delegate their votes to anyone. All root votes must be accompanied by the term, epoch, and ID of the delegation. The root leader tracks votes as follows:

- If the delegated vote is not in the current epoch, it is discarded
- Otherwise the leader accepts only the vote with the highest term (e.g. the current subquorum leader).
- If the leader receives a vote from the replica, then it accepts that vote no matter what.

In this way, it is possible for duplicate votes to be received by the leader, but only a single vote per replica will be counted.

Assassination

Scenario:

System size of 18, HC 5x3 (5 subquourms of size three, 3 hot spares) in 3 regions - 6 replicas per region. In epoch e there is a configuration such that all 5 subquorum leaders and the root leader are in a single region, which loses it's connectivity to the rest of the world. All followers have

delegated their votes to their subquorum leaders, except the hot spares which haven't delegated to anyone.

Just a note here, we've been calling it "assassination" but it's actually very difficult to create a scenario where this occurs if replicas are evenly distributed over all regions without violating the principle of localization. In the above example, every single subquourm is geo-replicated.

6 replicas with 16 votes can communicate and 12 replicas with 2 votes cannot.

Step one: all subquorums elect a leader for the next term; they continue to handle requests.

Step two: group of 6 attempts an epoch change to repair the poor progress, they assign all work to two subquorums of 3 (best scenario if there is total failure).

Step three: group of 12 attempts a root leader election, it fails because they only have 2 votes.

Step four: group of 12 initiates nuclear option, taking back delegations, moves into an epoch after the group of 6.

We now have the scenario where we have concurrent writes to objects in two partitioned regions, however because of the epoch ordering, we know that all writes that happen in the group of 6 happen before the writes that happen in the group of 12.

The group of 6 cannot make another epoch change because their delegations have expired and they cannot reacquire them.

Implementation and Evaluation

- Go implementation
- AWS EC2 across 15 regions and 150 replicas (3, size 3 subquorums in 15 regions with 15 hot spares).
- LevelDB
- Version numbers

Evaluation

Conclusion

- TODO: Psuedo code to make paper clearer
- **Bobby: Provide top down description, where each model is relatively small, reduce cognitive load**
- **Bobby: Change name "nuclear option" – make more technical e.g. "final recovery" (too jocular)**

References

[1] M. Biely, Z. Milosevic, N. Santos, and A. Schiper. S-paxos: Offloading the leader for high throughput state machine replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium On*, pages 111–120. IEEE.

[2] L. J. Camargos, R. M. Schmidt, and F. Pedone. Multicoordinated paxos. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 316–317. ACM.

[3] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, and others. Spanner: Google's globally distributed database. 31(3):8.

[4] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. MDCC: Multi-Data Center Consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126. ACM.

[5] L. Lamport. Fast paxos. 19(2):79–103.

[6] L. Lamport. Paxos made simple. 32(4):18–25.

[7] L. Lamport, D. Malkhi, and L. Zhou. Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, pages 312–313. ACM.

[8] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou. Boxwood: Abstractions as the Foundation for Storage Infrastructure. In *OSDI*, volume 4, pages 8–8.

[9] J. MacCormick, C. A. Thekkath, M. Jager, K. Roomp, L. Zhou, and R. Peterson. Niobe: A practical replication protocol. 3(4):1.

[10] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building efficient replicated state machines for WANs. In *OSDI*, volume 8, pages 369–384.

[11] I. Moraru, D. G. Andersen, and M. Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 358–372. ACM.

[12] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319.